SpaceOps-2023, ID # 324

Automatic And Adaptive Event-Based Observation Execution Engine Onboard the European Stratospheric Balloon Observatory

Mahsa Taheran Vernoosfaderani[a]*, Philipp Maier[a] , Andreas Pahler[a] , Sarah Bougueroua[a] , Sabine Klinkner[a]

[a] *Institute of Space Systems, University of Stuttgart, mtaheran@irs.uni-stuttgart.de*
* Corresponding Author

Abstract

With technological advances, balloon-borne telescopes are becoming a viable complementary platform for astronomical observation, along with space-based, aerial, and ground telescopes. More-reliable balloons and long-duration flight routes have led to an increasing number of initiatives aiming at more regularly flying missions rather than the more common "experiment"-type of flights. The European Stratospheric Balloon Observatory Design Study (ESBO DS), funded by the EU Horizon 2020 program, is one of these initiatives. ESBO is developing an infrastructure for regular astronomical observations from the stratosphere, creating a new operational concept for stratospheric science missions. The goal of ESBO is to improve the way scientific balloons have been used up to now, by creating an autonomous, highly flexible, and reusable platform, capable of integrating different instruments, long autonomous flights, and frequent launches. The promise of reusability and exchangeability of the instruments at the core of ESBO requires a flexible design both at hardware and software levels. ESBO would be providing an operating institution that offers observing time and instrument space on balloon-based telescopes for certain spectral regions, prominently including the mid- to far-infrared as well as parts of the ultraviolet.

The focus of this paper is the observation scheduling and execution onboard ESBO platforms. Without an automatic observation execution onboard, loss of communication and commandability may result in loss of observation time and science, reducing cost-efficiency of the platform. Some of the recent balloon missions, including PoGO, SUNRISE, and PILOT have successfully implemented some level of onboard autonomy in observation operations, using time-based execution or sequential execution of scripts. Time-based observations are also used in space telescopes such as Hubble. However, with changing instruments, observations performed from ESBO can vary from mission to mission. Therefore, time-modelling of each mission to plan the operations in advance could become time-consuming and labor-intensive, leading to difficulties in regular launches promised. In addition, time-modelling of stratospheric observation is not ideal, as the flight route is not strictly controlled and predictable. A strictly time-controlled approach cannot react to anything occurring during the mission, such as temporary decrease in stability or instrument failures, potentially rendering single observations useless. Therefore, ESBO uses a similar strategy to the James Webb Space Telescope, implementing an event-based automatic observation execution engine as part of its centralized flight software. This engine can handle observations for any instrument, as long as the instrument is integrated with the ESBO platform. Besides releasing commands to the different components for observation operations, the engine can detect failures and automatically performs contingency procedures. The file-based operations of the engine allow for easy modification of the observations during the flight. It also assists instrument developers in detailing the instrument operations by providing an easy-to-understand abstraction to define any generic balloon-based observation. In a long-term perspective, the event-based automatic execution engine will also open up the possibility to do on-board "success" checks of recorded data (in addition to more obvious failures) and to react to them (e.g., by checking general parameters such as signal to noise).

This paper describes the challenges of developing a science observation scheduling system for such a platform, reviews the requirements and constraints to consider and where they deviate from space and ground observatories, and presents an abstraction for balloon-based observations. It then focuses on the automatic observation execution engine onboard ESBO, detailing its architecture, major features, input files, and how it communicates with the rest of the flight software. Finally, the paper describes possible operational scenarios for ESBO observations, with different communication link availabilities to demonstrate how this engine and ground operator together control the telescope.

Keywords: Scientific Ballooning, Operations, Onboard Scheduling, Onboard Software

Acronyms/Abbreviations

COTS	Component off-the-shelf
UV	UlitraViolet
NIR	Near Infrared
MCP	MicroChanne Plate
ESBO	European Stratospheric Balloon Observatory
FSFW	Flight Software Framework
Json	JavaScript Object Notation
PUS	Packet Utilization Standard
ECSS	European cooperation for Space Standardization

1. Introduction

Scientific balloons can cater to a partly overlaying, partly different parameter space of mission requirements compared to space telescopes. Nevertheless, individual flights are considerably shorter than space missions, and depending on the exact wavelength, some atmospheric absorption may still exist. In addition, particularly interesting for infrared observations, cooling of large structures, e.g., telescope mirrors, to cryogenic temperatures is much more difficult than in space.

However, space observatories are intrinsically expensive and bear operational limitations: development times are long, updates or corrections of the instrumentation are usually not possible after launch, and consumables, such as cryogenic coolant fluids, cannot be refilled or replaced, as seen with the Herschel Space Observatory. Furthermore, rather conservative approaches towards new technologies are used to minimize risks of expensive failure. This is where balloons gain a lot of advantage with their less risk-adverse approach, and lower mission costs. This in turn, allows shorter development times, connected with the possibility to fly more up-to-date instrumentation. Regular returns of instruments furthermore allow their update and re-arrangement. This includes the possibility to refill cryogens, which is of particular interest for infrared observations. Furthermore, balloons come with less solid restrictions to the geometrical size of payloads, which are basically suspended in free air underneath the balloon.

Given the lower required investments and smaller timescales, balloons allow more flexibility and adjustability for specific scientific interests. Therefore, with the improvement of balloon technology, several balloon-astronomy initiatives are now aiming at more regularly flying missions rather than the more common "experiment"-type of flights. Noteworthy recent examples are the U.S./Canadian Super pressure Balloon-borne Imaging Telescope (SuperBIT) [1], the JPL-lead Astrophysics Stratospheric Telescope for High Spectral Resolution Observations at Submillimetre-wavelengths (ASTHROS) [2], and the U.S.-lead Balloon-borne Large Aperture Submillimetre Telescope (BLAST) [3] along with its successors.

The European Stratospheric Balloon Observatory (ESBO) is also aiming to take advantage of this aspect of balloon-based telescope. The modular ESBO platform can cater to the needs of different instruments or even different telescopes and provide regular flight opportunities for the science community. While even many of the above projects are still instrument-centric in their design, the ESBO platform is focusing on what can be made flexible and what can be a common module to reuse among different observation missions. The same approach takes the lead in the design of telescope control software. Most of the control software is instrument-agnostic.

This paper describes the observation execution as part of the onboard control software. The next section provides a short review of observation execution in space and stratospheric telescopes. To give more context to the design of software, section 3 briefly describes ESBO platform and the flight software, built for the first UV demonstrator, called STUDIO. Section 4 explains the design of the observation execution engine. Finally, in section 5, some improvements to upgrade the system are introduced.

2. Observation scheduling in space and stratospheric telescopes

Most earth-orbiting systems, including space telescopes, use time-based commanding in operations. In this approach sequence of commands with absolute, and occasionally relative, start times are uplinked to the system. Commands are stored onboard and then released and executed when the start time arrives.

The time-based commanding is used in Hubble Space Telescope [4]. The pre-requisite for such an approach is the possibility to precisely model the operations, the start time of commands, and the time duration of each, In the absence of such modeling, commanding conflicts are inevitable. For the systems that have longer operation times without ground contact opportunities, or those whose flight regime makes this modeling difficult, time-based approach is not ideal.

James Webb Space Telescope benefits from an event-driven commanding [4]. JWST schedules are uploaded to the flight system in the form of text-based scripts. These scripts issue observation commands to the flight software and check telemetry values to acknowledge the successful execution of the commands. Release of the next command is dependent on the successful execution of previous one, rather than any time-tag. The script can also skip an observation in case of faults and errors. Fig. 1. shows the high-level architecture of the system.

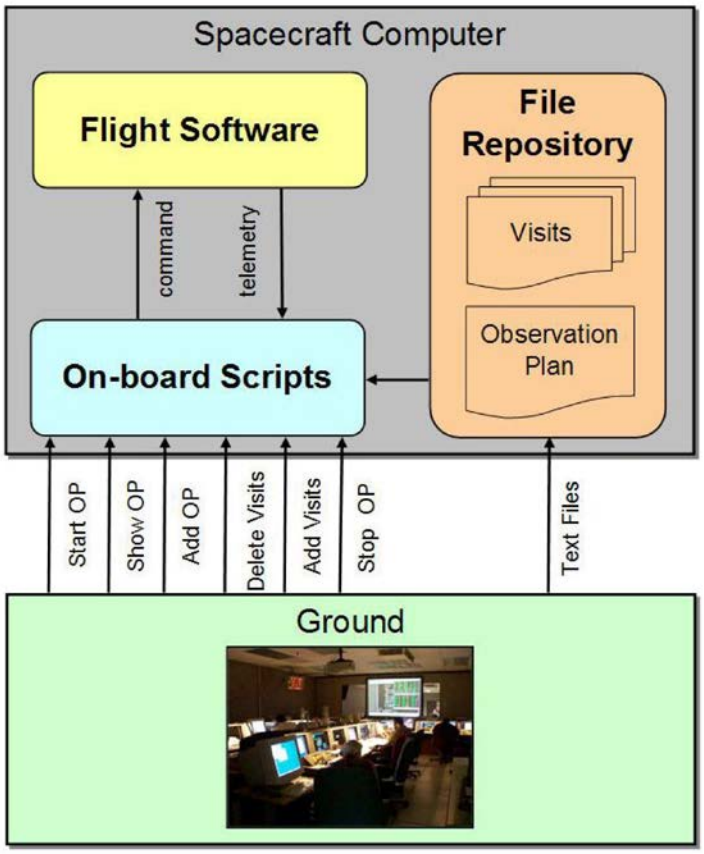

Fig. 1. James Webb event-driven observation execution[4]

Short term balloon-borne missions commonly benefit from a Line-of-Sight balloon to ground communication, allowing the operators of the system to command the system in real time and monitor the system simultaneously. Nevertheless, on longer flights this might not be the case, implying that we might need an onboard automatic mechanism for the task. Balloon missions, such as PoGo [5], Sunrise[6] and PILOT[7], have implemented different ways to handle observations autonomously, and to avoid observation time loss in case of communication failure.

They mostly use onboard fixed pre-defined scripts for autonomous observations. Observation scripts are usually defined as a parametrized fixed sequence of activities, based on relative time, that are performed for different areas of interests or targets. The parameters change depending on the target. PILOT and PoGo can also modify the sequence of the observations based on the time and location of the balloon. What is important is that the implemented autonomy is still instrument-centric and time-based.

The problem with time-based commands in stratospheric observations is that there is no logical time-dependence in the observation, and the activities such as pointing. Stratospheric observations are neither time-based nor position based. We can only know the time window during which a target may have (better) visibility. We can have estimates of how long pointing might take in ideal conditions, but it strongly depends on stratospheric environmental condition, wind, and perturbations. The perturbation models are limited. There is also no guarantee that a specific observation is successfully performed at a specific time, as it is not possible to certainly know how long it would take to do so. Failure of one task would therefore lead to a sequence of failures in the commands following up.

It would require quite some simulations to estimate how long it takes for each step of a full observation, from target pointing to the detector(s) operations. These simulations and observations are strictly mission-dependent and would change from one mission to another. Time-based scheduling of stratospheric observations is far from ideal, unless the system has flown many times and the behavior of all elements are known to a good extent. The disadvantages of time-based scheduling become more critical as the contact times with the flight system decreases, and the control over onboard schedule and the possibilities to manipulate and maintain the schedule on ground reduces.

3. ESBO Platform Design

The first ESBO-DS platform is a technology demonstrator with a UV detector and two science objectives:

- Search for variable hot compact stars
- Detection of flares from cool dwarf stars

This platform comprises a versatile, modular gondola for astronomical applications, carrying a 0.5 m aperture telescope suitable for observations in the UV to near infrared (NIR) wavelength ranges. Fig. 2. shows the telescope. The main scientific instrument is an advanced photon counting, imaging microchannel plate (MCP) detector. Complementary to the UV detector, STUDIO carries a scientific camera for the visible spectrum, that acts both as a scientific instrument and as the main sensor for the stabilization system of the telescope.

Fig. 2. STUDIO telescope in cleanroom

The gondola, shown in Fig. 3. is designed from COTS aluminum trusses in combination with custom-made tubing structures and holds the payload and all the subsystems. The modular design of each part allows for fast and

easy assembly and disassembly, with the possibility to house different, also larger, payloads (telescopes). The floor, where most of the electronics and service systems are located, uses fastener rails that offer multiple mounting points for more flexibility of positioning of the said systems. The gondola offers a coarse stabilization system reaching a stability of ~ ±40 arcsec in both elevation and azimuth. The fine image stabilization system placed on the Telescope Instrumentation Platform (TIP) compensates for the remaining jitter and achieves ±1 arcsec pointing stability.

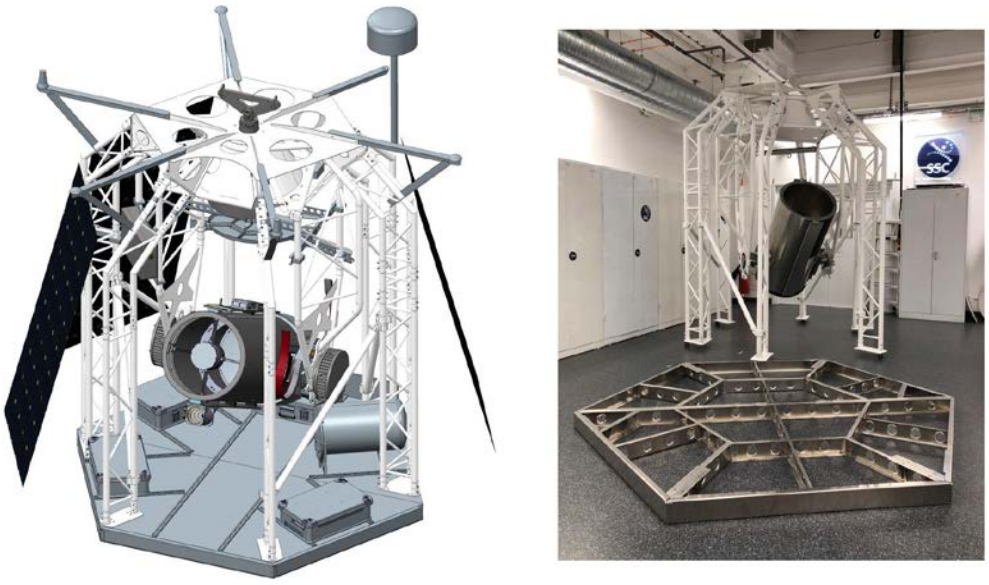

Fig. 3. The left graphic shows the STUDIO gondola. One solar panel is not shown for better visibility of the payload. Most electronics and service systems are located on the gondola floor. The right picture is the gondola integrated with the telescope dummy.

STUDIO's gondola can be adapted to fit larger telescopes. The pointing system and all electronics and service systems are also designed modular to cater to the needs of new instruments. The project has also designed concepts for balloon-based FIR telescopes with larger apertures for future construction and flights. These concepts are explained in Maier[8]. More information on current status of the STUDIO implementation and flight preparation can be found in Bougueroua[9].

3.1. Flight Software Design

Fig. 4. shows a high-level architecture schematic for onboard and ground computations of ESBO-*DS*. The onboard computation is divided into three major computers, communicating with each other through a UDP network onboard. In addition to these three computers, there are smaller platforms for more isolated tasks, such as the star tracker computer and the frame grabber card, which performs centroiding for the stabilization system.

The schematic also shows the ground segment elements. Most of commanding and monitoring activities are performed by a generic Mission Control System from OHB Sweden, called RAMSES, which is used for balloons and sounding rocket missions at Swedish Space Corporation. The choice is in line with the need for flexibility in architecture, as RAMSES can be used for any mission easily, as long as the configuration is added to the database correctly. Two other ground tools are developed to handle the science data pipeline and supervise the telescope pointing on a sky map, due to criticality of sun/moon avoidance angles. The payload blocks are all highlighted dark green.

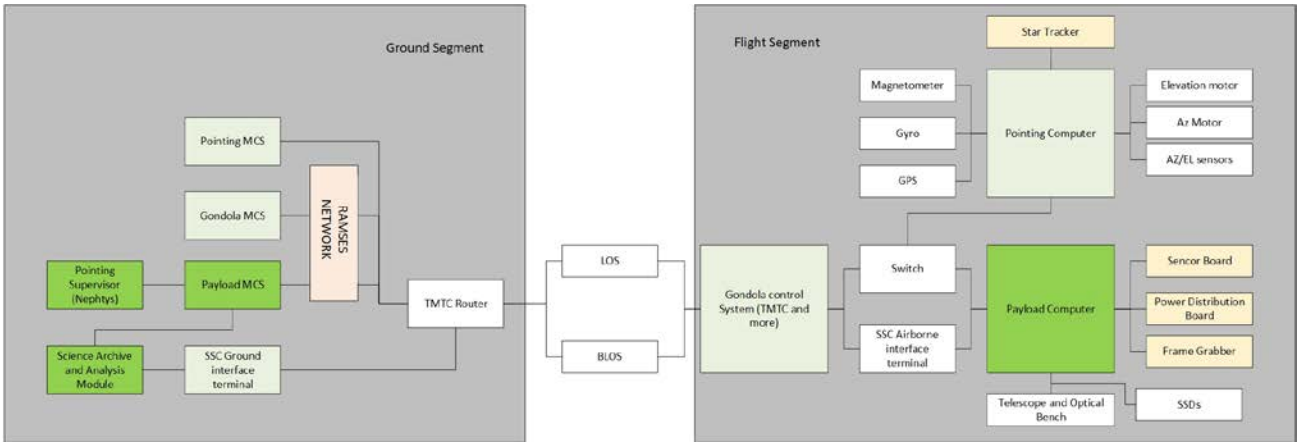

Fig. 4. ESBO-DS platform's computational architecture. Dark green blocks belong to payload, while the light green blocks are part of gondola/pointing system.

The driving requirement for the on-board software of ESBO is to be able to accommodate different instruments for different flights, and to fly regularly, integration of the mission-specific instruments in the software should be facilitated. Almost all payload related software onboard the balloon is integrated in payload flight software. This has a major architectural consequence, that instrument(s) control software should ideally integrate with the centralized payload software, or at least be controlled by the core software. While the operations are controlled centrally, the instrument(s) computations theoretically can still run on a separate platform.

3.1.1. Flight Software Framework

The on-board software for STUDIO is based on the Flight Software Framework (FSFW) that was developed for the Flying Laptop satellite and has a proven history of flight since 2018. The first version of the framework, developed at the Institute of Space Systems (IRS) is licensed open-source, and is currently used in other small satellite missions of the institute as well. For an in-detail paper on the FSFW please refer to Baetz [10].

While the FSFW has a steep learning curve and is not a black box framework, it provides the following advantages:

- Possibility to reuse most of the existing software for each mission. Changing the instruments, the telescope, or changing the service systems, would not require any changes in the core functionalities.
- Encapsulating most of the changes in clearly defined components.
- Facilitating integration of new software components, aka new instruments, for future missions, and the possibility of reusing the already implemented data exchange interfaces for new instruments.
- Having platform-independent, and OS independent software.
- Mode-based operations
- FDIR processes onboard
- Robustness of space standards

The FSFW is a component based object-oriented framework developed in C++, which implements common functionalities of the space flight software domain. Mission specific components connect to the FSFW by implementing specific interfaces.

4. Event-based Observation Execution

Given the spectrum of possible changes in payload, any onboard mechanism designed for automatic observation execution should be to some extent generic and configurable. It should also interface or integrate with the FSFW.

ESBO uses the Event-based approach to automatic observations onboard. The pre-flight planning process on the ground essentially results in a sequence of target stars to observe, with the corresponding observation variables, such as duration, instrument, and system settings, etc. Each observation might also have a corresponding observation window (earliest and latest start times to include tolerances), and possibly relevant system constraints that are required to be met for this specific observation. The designed automatic observation process onboard the ESBO-*DS* payload computer uses json files as input. Each observation is stored as a json file onboard, with a master plan which lists a sequence of the observation files to execute.

4.1. Observation model

Fig. 5. shows the abstraction defining an observation. In this generic mode, each observation consists of several phases that follow each other. Each phase can have one or more steps. The observation json file defines all the steps and phases for a specific target.

Steps are the basic building blocks of observations, each of which can be translated to one command to mission components within the software. Figure 10 shows the defining attributes of a step. These commands can be of three types:

- Mode commands setting a component mode.
- Action commands that command a device to perform an action.
- Parameter commands that set the value of a configuration parameter in a component

Steps can have contingency plans to be executed in case of failure. Contingency is also defined at phase level and observation level, indicating what should be done in case a step, phase, or observation fails. The contingency measures include ignore, repeating the step/phase/observation N times, or to consider observation failed and move to next observation.

Contrary to phases, steps can in fact run in parallel, or depend on specific previous steps. Phases are a higher-level building block. They do not run in parallel. They follow a specific logical sequence: for stratospheric observation these phases can be pointing, focusing, acquisition, observation take. Similar to steps, phases also have contingency measures. Failure of a phase is not monitored; it is derived from the failure of its steps.

On the top of the hierarchy, there is the observation itself. Observation as well has the contingency attribute. It also has a time window.

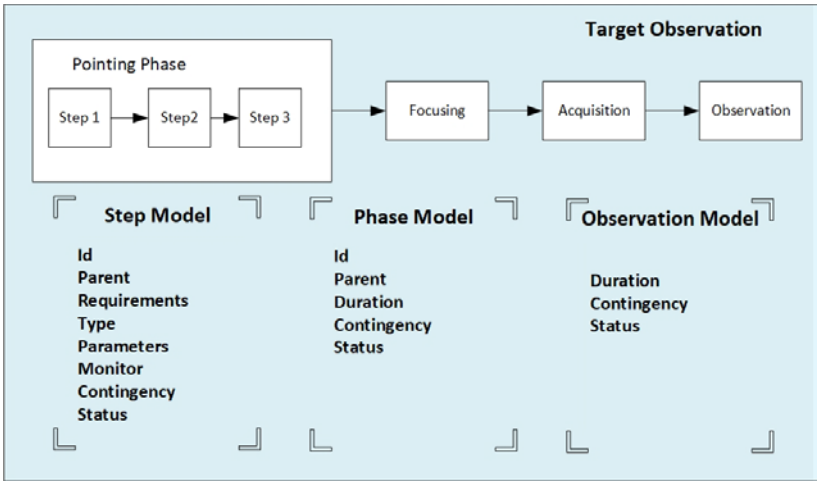

Fig. 5. Observation Abstraction used for ESBO.

4.2. Observation Executor

Observation Executor is a mission component, and is integrated with the FSFW, as described in section …..It can therefore easily communicate with all other mission components through the FSFW.

Observation Executor parses the observation files one by one, by checking the sequence of file paths in the master plan. The executor creates an observation object based on the file. As said, each step is a command. As the flight software implements ECSS PUS, the executor creates PUS commands and sends it to the target component.

4.3. Event feedback

FSFW implements an event management system, with publisher-subscriber pattern. Each mission component can issue events to the centralized event manager, and each mission object can subscribe to receive events from other components. Rather than checking housekeeping values, these events are used onboard ESBO to signal the execution feedback.

Each step in an observation has a monitor, that is the event to be observed. Observation executor subscribes to these event monitors and uses them to assert that a step has performed successfully. The process is depicted in Fig. 6. .Each step also has a timeout duration. If the success event is not received within this time, the step is considered failed, and contingency procedures are performed. As explained before, contingency procedure can be different for each step, ranging from full ignorance for non-essential steps, to repeating the steps, to skipping the observation fully.

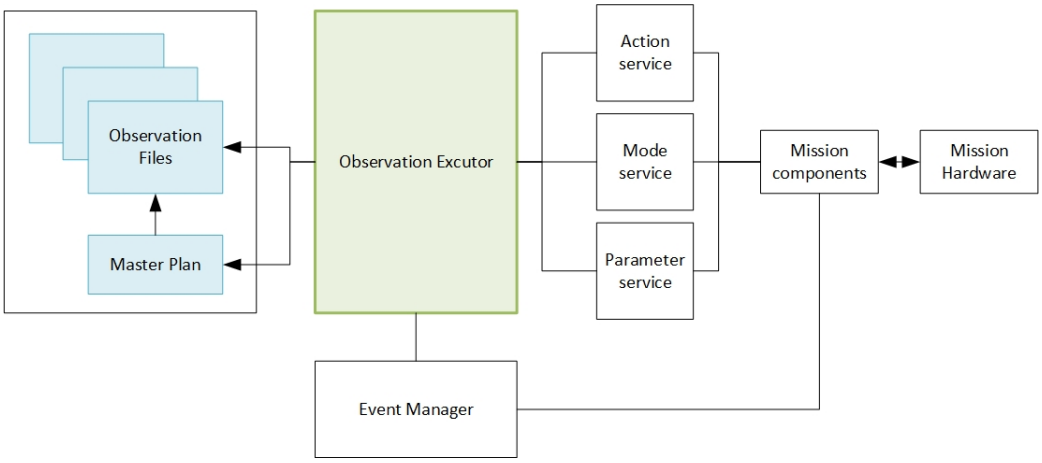

Fig. 6. ESBO automatic observation execution loop.

4.4. Automatic Observation Execution

Like any mission component, observation executor as well has its own modes. The observation executor automatically releases the observation commands as soon as the executor is in active mode.

There are two conditions that activate the Observation Executor: ground operator command, and loss of communication link. As the loss of communication is sensed onboard, the executor takes over control of the operations onboard. In a similar manner, the executor is deactivated on two conditions: ground operator command and system fall to safe mode, as described in Fig. 7.

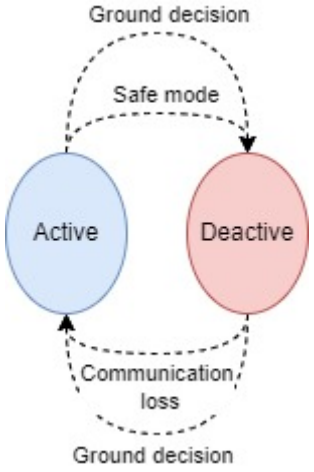

Fig. 7. Observation executor modes

4.5. Operational Scenarios

With the implemented observation execution, the platform is capable of operating in two different regimes:

- Operator-commanded observations

The simplest case would be to have constant real-time access to the system. This is the ideal situation, and as long as the communication to ground is available, can be used by the operator. This regime is still event-based. The operator observes the telemetry data to assert the successful execution of each command, before proceeding with the next steps.

Given the bandwidth in LOS communications and processing of downlinked scientific data, the observation plans can be modified and the operator can adjust the commands and instrument settings.

In BLOS communication, the system is still commandable. In principle the operations follow the same process and the ground operator has full control. What is limited in this regime is the observability of the system. STUDIO software is designed to downlink the critical feedback and major housekeeping data first in this communication mode, but the scientific data downlink is limited and may not be used to modify and adjust the observation settings.

- Autonomous observations:

In absence of communication links, autonomous observations start. In this regime, and as explained above, onboard observation files are executed. At this stage these pre-defined observations are static and not modifiable. The settings are also not adjusted onboard.

5. Improvements

There are two features that can be added to upgrade the first version of the observation engine.

5.1. Schedule Modifications

File-based operation allows the easy replacement of observation files onboard. The master plan file includes a sequence of paths. It is possible to replace the master file to change the sequence or to update it with new observation files. This is an essential feature for a longer duration flight. This process requires a file transfer protocol to be implemented onboard and on the ground.

5.2. Observation Tracking

Observations may transfer between automatic and operator-commanded modes several times during a flight. It is therefore essential to make sure the sequence of observations is tracked on each mode and the current/next observation to execute is always known to both.

6. Summary and Future

The flight and ground software of the European Stratospheric Observatory is designed with one central objective: to be flexible enough to integrate different instruments and be used in different missions for various observations. The core Flight Software Framework provides the needed extensibility for integrating new instruments with the payload software.

The previous sections described the approach used for automatic observation execution onboard. This approach reduces the need for time-modelling for each flight and simplifies the whole scheduling process for observations. It also is a way to better use the flight time. In addition, this also facilitates the planning process with instrument scientists, as the human-readable files are considerably easier to discuss rather than binary commands.

STUDIO hardware development is in its final stages, and qualification tests of the telescope and optical bench have been successful. In addition, the telescope inner stabilization system has also been successfully verified. With most of the software components implemented, flight software integration should follow soon. The science and telemetry downlink, and ground software components have been verified in lab tests with simulated images.

While the first flight of STUDIO would validate the approach laid out here to some extent, flying with a new instrument afterwards can demonstrate if the software positively affects the integration time in practice.

References

[1] A. Cho, Cheap balloon-borne telescopes aim to rival space observatories, Science Magazine, 27 February 2020, https://www.sciencemag.org/news/2020/02/cheap-balloon-borne-telescopes-aim-rival-space-observatories, (accessed 15.01.2023).

[2] C. Cofield, NASA Mission Will Study the Cosmos with a Stratospheric Balloon, NASA, 23 July 2020, https://www.jpl.nasa.gov/news/news.php?feature=7712 (accessed 15.01.2023).

[3] N. Galitzki, *et al.*, Instrumental performance and results from testing of the BLAST-TNG receiver, submillimeter optics, and MKID detector arrays, Proc. SPIE 9914, 99140J (2016).

[4] I. Dashevsky, V. Balzano, James Webb Space Telescope Ground to Flight Interface Design,2008 IEEE Aerospace Conference, Big Sky, MT, USA, 2008, pp. 1-7, doi: 10.1109/AERO.2008.4526655.

[5] M. Chauvin, HG. Florén, M. Jackson, *et al.* The design and flight performance of the PoGOLite Pathfinder balloon-borne hard X-ray polarimeter. *Exp Astron* 41 (2016), 17–41.

[6] A. Bell, P. Barthol, *et al.*, Flight control software for the wave-front sensor of SUNRISE 1m balloon telescope, Proc. SPIE Vol. 7740 (2010).

[7] M. Saccoccio, J.P. Bernard, *et al.*, Operations and results of the PILOT balloon borne telescope flight, SpaceOps Conference, Daejeon, Korea, 2016.

[8] P. Maier, M. Ångermann, *et al.*, Stratospheric Balloons as a Complement to the Next Generation of Astronomy Missions ,71st Interntational Astronautical Congress, 2020, 12-14 October.

[9] S. Bougueroua, M. Ångermann, *et al.*, Status, Flight Preparation, and Future Instrument Opportunities of the STUDIO Balloon-Borne Telescope Platform, SPIE Astronimcal Telescopes+Instrumentation, Montreal, Canada, 2022.

[10] B. Baetz, U. Mohr, K. Klemich, N. Bucher, S. Klinkner, J. Eickhoff, The Flight Software of Flying Laptop: Basis for a reusable Spacecraft Componente Framework, IAA Symposium, Berlin , Germany, 2017.